Please cite this paper as:

Richard A Williams (forthcoming), User Experiences using FLAME: A Case Study Modelling Conflict in Large Enterprise System Implementations, Simulation Modelling Practice and Theory

ACCEPTED FOR PUBLICATION | September 16, 2020

ORCID NUMBER: 0000-0001-6333-9448

DOI: Not yet Assigned

Dr Richard Alun Williams
Senior Lecturer in Management Science
Lancaster University Management School
Lancaster, LA1 4YX

User Experiences using FLAME: A Case Study Modelling Conflict in Large Enterprise System Implementations

Richard A. Williams

Department of Management Science, Lancaster University, Lancaster, UK

Abstract

The complexity of systems now under consideration (be they biological, physical, chemical, social, etc), together with the technicalities of experimentation in the real-world and the non-linear nature of system dynamics, means that computational modelling is indispensible in the pursuit of furthering our understanding of complex systems. Agent-based modelling and simulation is rapidly increasing in its popularity, in part due to the increased appreciation of the paradigm by the non-computer science community, but also due to the increase in the usability, sophistication and number of modelling frameworks that use the approach. The Flexible Large-scale Agent-based Modelling Environment (FLAME) is a relatively recent addition to the list. FLAME was designed and developed from the outset to deal with massive simulations, and to ensure that the simulation code is portable across different scales of computing and across different operating systems. In this study, we report our experiences when using FLAME to model the development and propagation of conflict within large multi-partner enterprise system implementations, which acts as an example of a complex dynamical social system. We believe FLAME is an excellent choice for experienced modellers, who will be able to fully harness the capabilities that it has to offer, and also be competent in diagnosing and solving any limitations that are encountered. Conversely, because FLAME requires considerable development of instrumentation tools, along with development of statistical analysis scripts, we believe that it is not suitable for the novice modeller, who may be better suited to using a graph-

Email address: r.williams4@lancaster.ac.uk (Richard A. Williams)

ical user interface driven framework until their experience with modelling and competence in programming increases.

Keywords: Agent-Based Modelling, Complex Systems, Computational Social Systems, High-Performance Computing

1. Introduction

Over the past two decades, the use of computational approaches to investigate social systems, including organizational and behavioural research has progressed from mere quantitative data analysis to one that complements traditional social science techniques through the use of advanced techniques in modelling and simulation. The techniques used within computational social science have evolved from mathematical approaches that utilize deterministic or stochastic equations, through to computational modelling approaches such as agent-based modelling and simulation [1, 2]. An advantage of these computational modelling approaches is the ability to fully control the underlying mechanistic component interactions of the model, thus once a computational model is established, quick studies of the effects of changes to the elements and associated parameter values can be seen through equation-solving or simulation. As such, simulation-based experimental results from a well engineered model are directly related to the level of abstraction and assumptions made during model design and development. This provides a firm basis for testing hypotheses on the mechanisms behind social behaviours and dynamics than traditional approaches such as participant observations or action research, where the unseen variables/factors can introduce additional complexity and uncertainty, which limits our ability to translate interpretations of observational/qualitative results to the real-world system.

Real-world systems are complex, with a range of system behaviours, properties and characteristics that emerge from the low-level interactions of a highly connected set of system components that function through temporal and spatial dimensions. One of the main strengths of the systems approach to investigating complex systems, is that it focuses on three key properties when modelling the systems: 1) system structures, 2) system dynamics, and 3) system control [3]. Indeed, previous discussions around Agent-Based Modelling and Simulation (ABMS) advise that it can be used to test *working* theories of the underlying mechanics of component interactions within systems and their resulting dynamics [4, 5]. As such, we believe that ABMS is the preferred

approach to facilitate the micro-to-macro mapping of complex dynamical systems, in particular, social systems [6]. This is performed through altering the way in which agent interactions occur, or relaxing of assumptions at the individual agent-level, in order to investigate the emergence of system-level behaviours at the system-level. We therefore believe that ABMS provides a low-cost computational approach for developing hypotheses of the real-world system. With ABMS having recently become accepted as a scientific instrument for investigation [7], it is now imperative that agent-based models (ABMs) adhere to the scientific method through a principled approach to their design, development and use, to ensure they are fit-for-purpose [4].

Technology advances in hardware architecture, data storage and processing speed over the past twenty years has encouraged the development of new simulation frameworks that can handle increasingly complex computational models. With respect to ABMS, this has resulted in the development of a number of frameworks aimed at either subject matter experts or novice modellers, which use graphical user interfaces, or experienced computational modellers that are comfortable with command line terminals, a range of basic programming languages and shell scripting. A relatively new ABMS framework that falls in the latter category is the Flexible Large-scale Agent-based Modelling Environment (FLAME). Within this study we have used FLAME to model the development and propagation of conflict within a large multi-partner enterprise system implementation.

This manuscript provides an account of our experiences in using FLAME, with particular reference to a number of it's strengths and weaknesses with respect to: ABM design; ABM development; constraints due to its underlying technical architecture; and performance. In particular, we highlight: the ease of model design for experienced computational modellers who are familiar with object-oriented programming and the Unified Modelling Language (UML); and the ease of model development for experienced computational modellers who are comfortable with the logic behind UML State Transition diagrams. In addition, we discuss a number of significant constraints of FLAME due to the underlying architecture, such as: issues when setting the Pseudo-Random Number Generator seed value; the inability to send messages between simulation time-steps; the inability to use global mutable parameters; performance challenges due to the I/O rate-limiting characteristics and the need for significant fast storage allocation to accommodate realistic simulations of social systems. The manuscript is structured as follows: section 2 provides an overview of the major concepts that contribute

to the theory behind our study; section 3 discusses the case study; section 4 provides an account of our user experiences when using FLAME to model the case study; section 5 is the discussion section; and section 6 provides our conclusions.

2. Related Work

This section provides an overview of the major concepts that contribute to the theory behind our study.

2.1. Agent-Based Modelling and Simulation of Social Systems

Social networks and the various dynamics and emergent behaviours that develop from interactions between individual human agents are complex, meaning that they often display non-linear dynamics. Computational social scientists often model these social dynamics through equation-based mathematical approaches (e.g., system dynamics or differential equations) or ABMS. These computational modelling approaches facilitate the analysis and investigation of complex social systems that would otherwise be intractable. This may be due to direct experimentation of the system being either unethical or impossible across the scale and hierarchies of the social system; difficulties in observing system dynamics due to extremely short or indeed large timescales; difficulties in observing the whole population due to the magnitude of the system, or to the location or complexity of the system (e.g., virtual team).

Importantly, it is frequently apparent that a significant amount of data generated through observations, participant interviews or surveys in the social sciences, accumulates and is collected in a manner comparable to the object-oriented paradigm for design of computational systems. Historically, computational modellers have taken inspiration from the natural and physical sciences and utilized the reductionist approach, which advocates that systems should be investigated through their decomposition to their smallest indivisible unit, which we believe is analogous to looking for 'objects' within nature. As such, we believe that an object-oriented approach to computational modelling, through the use of a bottom-up approach for design and development of the models is a useful formalism for modellers in general, and those interested in Computational Social Systems in particular. ABMS intuitively expands upon the object-oriented approach to model design and development, and significantly benefits through the principal enhancement that an *agent* is active instead of passive, and secondarily, that ABMS is

not bound by a central computational mechanism of control, but instead has multiple threads of control. Macal and North [8] provide a tutorial on ABMS, which describes this by stating that the *"fundamental feature of an agent is the capacity of the component to make independent decisions"*. In addition, Jennings [9] further extends the notion of an agent by advising that they are asynchronous components of a computational model that interact with the environment. As such, agents have the capability to *sense* their environment, and through processing the input(s) that they receive from the environment, may act upon it.

The origins of ABMS are in the modelling and simulation of human societies, systems, and networks and the associated dynamics and relationships within these hierarchical structures [10], with particular connection to the fields of: Complex Systems, Computer Science, and Management Science [8]. In fact, the agent-based approach was developed to allow investigation of complex social systems at the level of individual actors/people, so borrowed the reductionist perspective from the Natural Sciences. The belief here, which was also borrowed from Engineering, is that complex systems are built from the ground up, which was a marked contrast to the beliefs of Systems Thinkers who took a top-down view. Since its inception, the agent-based approach has gained the attention of researchers from fields far away from its origins within the social sciences. Examples of the use of ABMS to model complex dynamical social systems include: the seemingly inocuous renting and sales patterns of houseowners/tennants within a reimplementation of Shelling's Bounded Neighbourhood Model [11]; the most recent banking crisis associated with the British banking sector [12]; the way that virtual project team performance is affected by communication technologies [13]; and the recent Covid-19 pandemic [14].

The behaviour of individual agents is dictated by rules for their behaviour(s) and interactions. The interactions between compatible agents gives rise to behaviour(s) and dynamics at the system-level. This population-level behaviour, develops through the aggregated dynamics of individual agents, and is consequently deemed an emergent behaviour because it is not defined in the technical design of the computational model. Epstein [6] advises that this study of the emergent system-level behaviours is one of the key advantages of the ABM approach, and the ability to investigate the effects of the heterogeneous and autonomous individual agents has led to its increased adoption over the past decade [8, 10]. Moreover, an agent-based approach allows the modeller to design and develop the model by explicitly taking

the individual agent's location into consideration, which results in the model more accurately reflecting the spatial aspects of the complex social system, and in particular the network of social interactions within the system.

The engineering of viable computational models of social networks is not a straightforward process. This is due to the real-world system containing multiple feedback signals and non-linear dynamics, along with the model requiring the use of numerous parameters whose values have a degree of uncertainty (or may even be unknown) from observations/participant interviews. This uncertainty is further compounded through the introduction of randomness at key decision points/system events to provide the heterogeneity in behaviours that are seen across a population of human actors within the system [15]. ABMS equips the model developer with a set of techniques to integrate multiple data types gained from empirical data collection and knowledge themes gained from secondary data sources. Furthermore, one of the key strengths of the agent-based approach is that it allows us to model the complex non-deterministic and heterogeneous behaviours, which allows us to investigate the role of differences in chracteristics (e.g., demographic or behavioural) of individuals within social systems and develop testable hypotheses for new empirical research. See Williams [4] for lessons learned on development and application of ABMs of complex dynamical systems.

The work of Nikolai and Madey [16] has reviewed the main agent-based modelling frameworks available, and discussed their different computational underpinnings; examples include MASON [17], Repast [18], and NetLogo [19]. The commonality between all of these frameworks, is that they represent the individual agent as having a specific *state* at any particular point in time. Consequently, representations of each individual agent may be considered as a *state machine*, where the combination of current and previous inputs (previous states), along with the logic/functions associated with the agent-type, determines the output (next state) of the agent.

Increasing progress in hardware architecture, storage and speed over the past twenty years has encouraged the development of new simulation frameworks that can handle increasingly complex computational models, which simulate at magnitudes and dimensions that are much more closely aligned to real-world societal populations. For example, the Flexible Large-scale Agent-based Modelling Environment (FLAME[1]) was developed [20] to provide an

[1]http://www.flame.ac.uk

intuitive development framework for large-scale ABMs. FLAME was developed for use over the full range of computer hardware, which allows for the early coding and unit/system testing on a personal computer, before being moved to high-performance computing infrastructure for full-scale testing and simulation. FLAME has been used by researchers across a variety of domains, ranging from Biology [21], to Economics [22], to Human Resources [23], to transport and logistics [24].

2.2. The FLAME Simulation Framework

The FLAME simulation framework comprises a collection of Application Programming Interfaces (APIs), templates to define the agent-based model, and compilation and parsing routines to create the C code for running simulations. Due to FLAME's conceptual architecture being based on communicating stream X-Machines, templates are used to define the rule-based functions (written in C code) of agents, with the agent types being defined through XML templates, and the instantiation of the initial iteration of a simulation run using another XML template that defines the attributes and internal agent states for the simulation run. FLAME requires installation and configuration of the Message Passing Interface (MPI) to develop messages that communicate interactions between specific agents and changes within the system's environment. These agent interactions are simulated through transition functions that specify the rule-based logic of the ABM, with all agents that transition to a new state having their internal memory updated accordingly. Following specification of the model in the appropriate templates (e.g., C and XML), the xparser tool is used to generate simulation code (see figure 1). The xparser is written in C with the use of standard libraries, and is actually a collection of template files and header files that are stored in the desired directory of the computer (i.e., laptop, desktop, or HPC) and compiled using the freely available GCC compiler. Furthermore, through the use of the MPI communication framework, the generated simulation code is also portable to HPC platforms that use parallel architectures, thus enabling efficient communication between agents that may be across different compute nodes, and therefore remain in sync [25].

An X-machine is a formalized specification developed by Eilenberg [27] that has the capability to model both the *system's* data and the specification method (*function*) for controlling the system. They were originally introduced in 1974, but did not receive widespread acknowledgement in the modelling community until 1988 [28] when the approach was advocated as

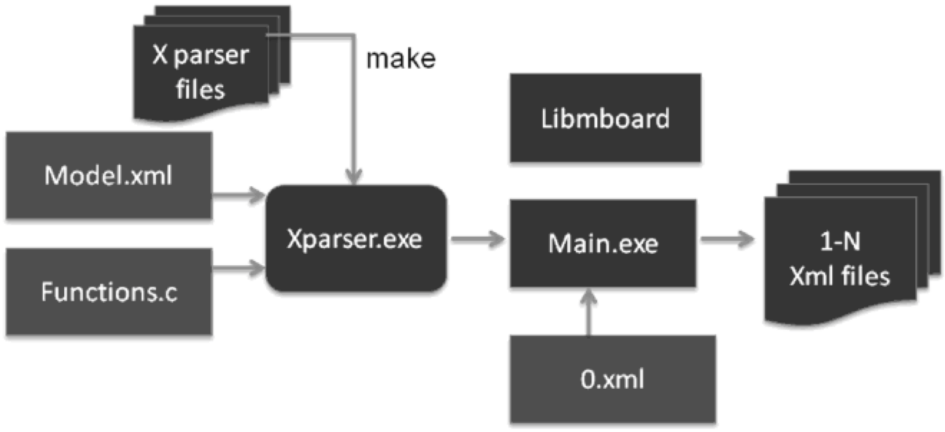

Figure 1: FLAME relies on the definition of models using the XML and C templates, and their parsing and subsequent compilation into simulation code. This simulation code is then run, through linking the main executable file to an initial starting parameters file (0.xml). Due to FLAME utilizing the discrete-event approach, individual simulation runs generate separate output files for each time-step increment within the simulation, with each output file acting as the input file for the subsequent simulation time-step (after [26]).

the basis for a formal model specification language. Subsequently, this was built upon in 1992 through the concept of stream X-Machines, which utilized sets of input and output symbols that were read through in a stream. From a conceptual modelling perspective, X-Machines employ both a diagrammatic approach and formal notation to model the system, where the X-Machine contains infinite internal memory, a set of functions, and a current state. The current state of control (the function defined in the specification method) and the current state of the memory, is processed alongside an input symbol from the input stream to determine the next state of the X-Machine, update it's memory state, and calculate the output symbol, which becomes part of the output stream that is used for communicating to other X-Machines. This can be summarized in that the system's new *state* is a product of it's current state (using memory and reference to a list of input symbols) and the relevant function [29].

A communicating stream X-Machine model is a formalized specification that builds upon that of X-Machines to introduce additional modelling capabilities. In particular, communicating stream X-Machines can be used to compute the functional behaviour of the system at a lower-level of granularity (i.e., individual agents), whose individual dynamics may be aggregated

up to the overall system level in order to generate the emergent behaviour of the system as a whole. The formal notation of the communicating stream X-Machine specification utilizes a 10-tuple notation, with C_i^X representing the ith communicating stream X-Machine component, which is defined by Stamatopoulou [30] as:

$$C_i^X = (\Sigma_i,\Gamma_i,Q_i,M_i,\Phi_i,F_i,q_{0i},m_{0i},I\Phi_i,O\Phi_i)$$

where:

- Σ and Γ are respectively the input and output alphabets.
- Q is the finite set of real-world system states.
- M is the (possibly) infinite set that relates to memory.
- Φ, the *type* of the X-Machine X, is a set of partial functions φ that translates a specific input and a specific memory state to a specific output and a possibly different memory state, $\varphi : \Sigma$ x M $\rightarrow \Gamma$ x M.
- F is the next state partial function, F : Q x $\Phi \rightarrow$ Q, which given a state and a function from the type Φ determines the next state. F is often described as a state transition diagram.
- q_0 and m_0 are respectively the initial state and initial memory.
- $I\Phi_i$ is the communication interface for the input messages.
- $O\Phi_i$ is the communication interface for the output messages.

An X-Machine is defined by Holcombe [28] as a system that has internal memory and an internal *computational* state, which dependent on environmental input and their current internal state, can transition to another computational state (see figure 2). The additional functionality incorporated into communicating stream X-Machines means that they are able to model in a single process specification, the functional behaviours and dynamics of an agent, as well as the intrinsic system data that it is modelling. A *communication matrix* using MPI facilitates communication (and thus interactions) between individual communicating stream X-Machines. This communication matrix is essentially a simulation-level *message board* that facilitates the reading and writing of information between every communicating stream X-Machine.

2.3. Enterprise System Implementations

The Gartner Group devised the term Enterprise Resource Planning (ERP) in 1990 [32] to define the software that provided functionality for organizations to use to manage their core administrative (back office) functions such

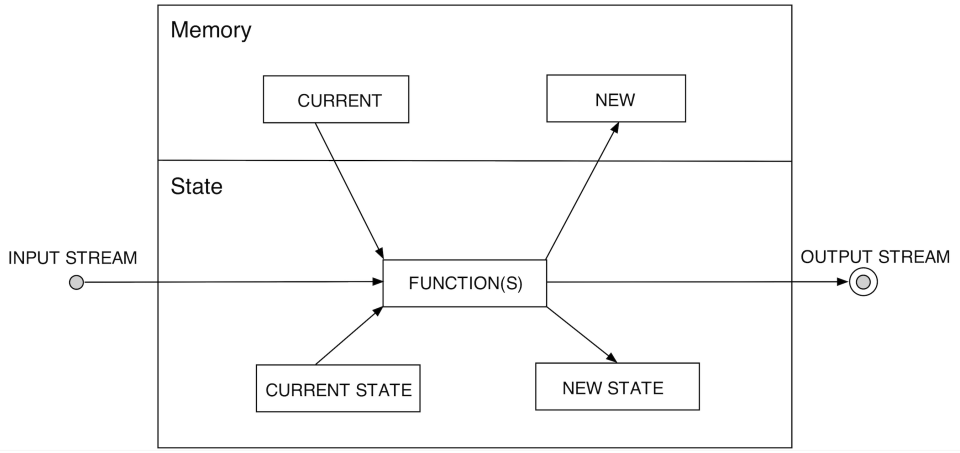

Figure 2: Diagrammatic representation of a generic communicating stream X-Machine. Relevant input messages that are held within the communication matrix are read by an agent, and potentially processed to initiate state transitions and updating of the agent's current memory and current state [31].

as Finance, Payroll and Human Resources. A number of leading software vendors (e.g., Oracle, JD Edwards, PeopleSoft and SAP) offered ERP software packages to store the organizational data and provide assistance to facilitate compliance to a *standardized* set of business processes [33]. With the expansion of Information Systems (IS) and Information Technology (IT) throughout the late twentieth century, the individual software modules that focused on the different organizational business functions were integrated together. These larger software systems, which were termed Enterprise Systems, not only integrated the individual ERP modules, but also provided additional functionality, such as business intelligence, advanced planning, and automatically processing data from external supplier/customer relationships. Enterprise systems are now commonly used to help organizations (of all sizes) manage their human, financial and physical resources (e.g., staff, money, and products) along with key external relationships (e.g., customers and suppliers) more effectively [34].

Enterprise systems require considerable time and resource commitments in order to be implemented properly within an organization. These implementation projects within large organizations are usually implemented by third-party service providers, with the largest implementations using the consultancy services of both the vendor (e.g., SAP or Oracle) and professional services firms (e.g., Accenture, IBM, CapGemini, etc.). The various

customer, vendor, and professional services personnel are usually structured into project teams that relate to the functional modules within the enterprise system, such as Finanicals, Payroll and Human Resources, which gives rise to a wider implementation programme.

Whitty [35] has postulated that the increasing complexity and scale of reach into the business by these enterprise system implementation programmes, contributes to them exhibiting similar behaviours and traits to those of complex systems. We have previously discussed that the emergent behaviour generated within the social network of these large implementation programmes, may, to a large degree, be due to the complexity stemming from the large number of team members, and the growing trend of using multiple third-party providers to implement the individual projects that combine to form the programme. In addition, we discovered that the individual project teams may have competing priorities and objectives, and that these may lead to various forms of conflict propagating throughout the wider programme, which we view as a social network of formal workplace relationships [36].

2.4. Conflict within Enterprise System Implementations

Conflict within group situations, such as the project teams that implement functional modules of enterprise system software, has been defined as interpersonal incompatibilities or the holding of divergent views/perspectives amidst the members/participants, which may be individuals within a single group (intragroup conflict) or between different groups (intergroup conflict) [37]. Conflict has been reasoned to be an intrinsic part of group [38] and project team dynamics, which can propagate throughout the network of group members as a shared affect [36]. Conflict develops during a variety of circumstances relating to the implementation of group- and team-based activities, and in three main forms: task, process, and relationship conflict [39].

Enterprise system implementations at large organizations, often require the personnel, expertise and resources of multiple third-party organizations, which may have different, and often incompatible, business objectives and commercial drivers. As discussed above, the programme-wide implementation of the enterprise system is routinely divided into discrete project teams that map on to the corresponding functional modules within the enterprise system software (e.g., Finance, Payroll, HR). Our recent study [36] showed that within large multi-partner enterprise system implementations, conflict (be that task, process or relationship) can develop between members of a particular project team, or between members of different project teams, and once

developed, can propagate throughout the social network of the multi-partner enterprise system implementation. Furthermore, this study conceptualized the propagation of conflict within large multi-partner enterprise system implementations as conflict propagating between team members of an individual project team, and also between team members that are situated within different project teams (see figure 3). In addition, we built upon the work of Gamero *et al.* [40] who discovered that conflict is dynamic and can transition over time, by hypothesizing that task or process conflict can transition to relationship conflict, and that relationship conflict (being directly related to shared affect), is the most common form of conflict to propagate through the social network of enterprise system implementations. We therefore hypothesize that the conflict, which initially developed between a subset of team members from two different project teams, has the potential to propagate throughout the social network of the wider enterprise system structure, and may negatively affect implementation of the enterprise system as a whole.

3. Case Study

The case study relates to a large UK-based organization that was implementing an enterprise system along with associated IT hardware as part of a major strategic modernization initiative. This IS/IT change programme was underpinned by the need to integrate the enterprise system with multiple, existing, third-party legacy systems, to facilitate significant cost savings and promote more efficient business processes. The client selected a large Enterprise Applications vendor to implement the installation and configuration of the enterprise system, and to lead the design and assist in the development of the technical architecture that the enterprise system was embedded within. In addition, two different Professional Service Providers were also contracted to act as Domain Expert Consultants for the configuration and extension of functional modules within the enterprise system, along with a third Professional Service Provider who consulted on the hardware and middleware requirements for hosting the enterprise system.

Such an environment is best described as a multi-partner enterprise system programme implementation. The resulting social network was structured into project teams that aligned to the functional modules within the enterprise system, along with a Programme Management Office (PMO) and project teams that focused on developing technical extensions (e.g., custom forms, reports, and database triggers), training material, and installing

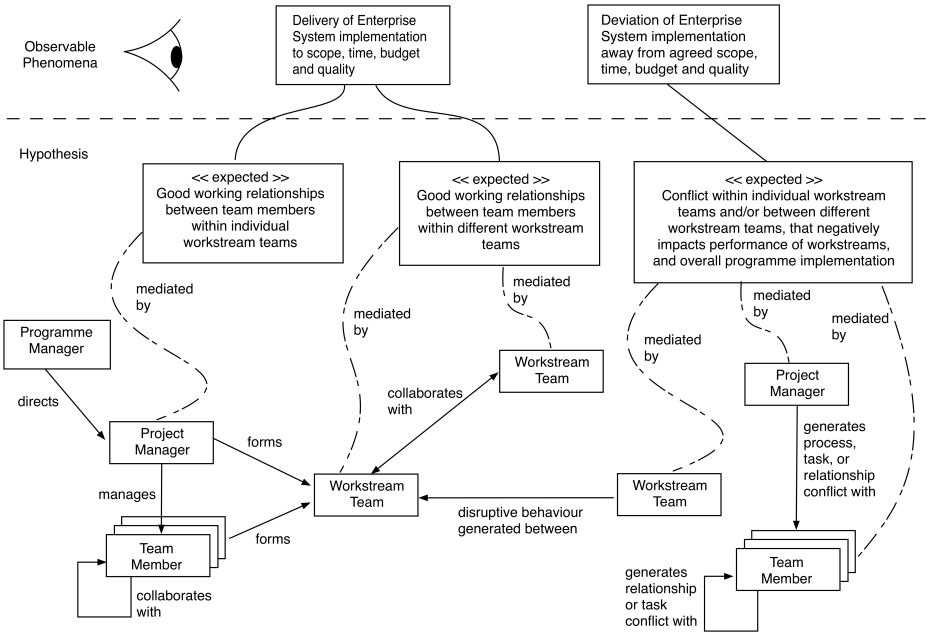

Figure 3: Rich picture: Observable phenomena that emerge from the interactions between individual team members. It is hypothesized that conflict can develop through a number of mechanisms, and that once formed, it may propagate throughout the social network of the wider programme. If any of these occur, they may be observed as deviations away from the agreed scope, time, budget and quality of the enterprise system implementation. Reproduced from [36] under Creative Commons Attribution 4.0 License.

and configuring the technical architecture that *hosted* the enterprise system. Overall, there were 159 resources from these five organizations, with 972 undirected workplace relationships between them, which formed the basis of the social network map (see figure 4). The Conceptual Framework presented in [36], acted as our conceptual model for the ABM, by providing the phenomenological behaviours that are the basis of the emergent behaviours generated from the ABM (see figure 3), alongside the detailed person-level data and social network map that formed the basis for the individual agents within our computational model, and the rules that dictate which agents can interact with each other.

The case therefore constitutes a large multi-partner enterprise system implementation, which utilizes IS/IT consulting personnel from four external organizations. These external resources were combined with client resources, to form a number of project teams that align to the underlying functional

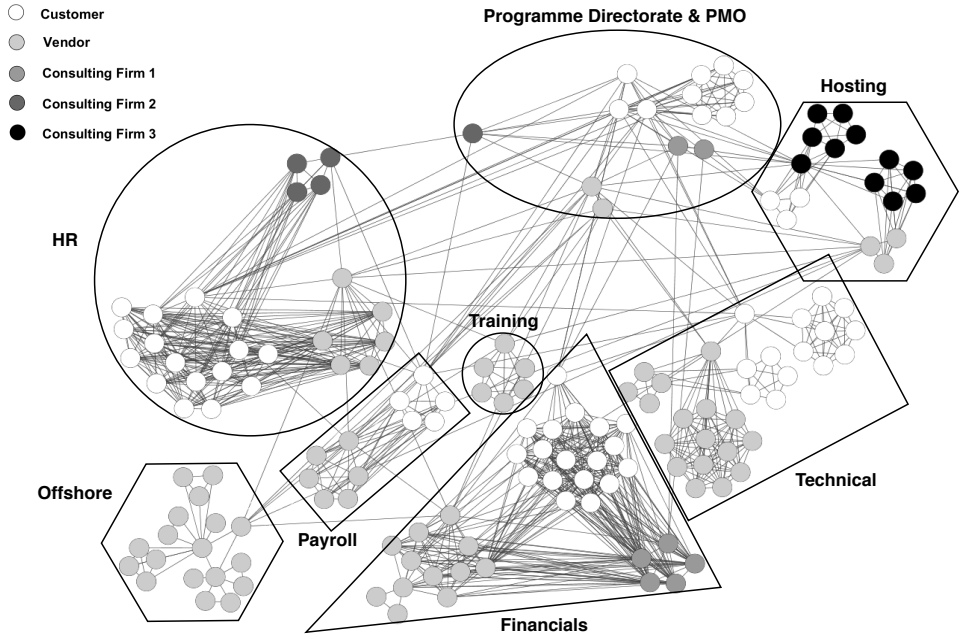

Figure 4: Social network of the case. The programme structure of the multi-partner enterprise system case study is represented by a social network map that defines how resources from the various organizations are assigned to the individual project teams. It can be seen that the individual project teams are structured around the functional modules within the enterprise system software (e.g., Financials, HR, etc.) and that they do not work in isolation, but instead require interactions with other project teams in order to implement their specific project objectives, and in turn facilitate implementation of the wider programme. Reproduced from [36] under Creative Commons Attribution 4.0 License.

modules of the enterprise system. The resources worked together within their respective project team, with a number of them also acting as *bridgers* to interact with other project teams, which is diagrammatically represented in figure 4 as the programme-wide interfirm social network. This multi-partner social network, introduces a risk that conflict may develop due to the differing backgrounds of project team personnel (e.g., different professional identities, cultural backgrounds, education, and normalized behaviours), along with their constrasting drivers and motives for being at their chosen employer (and occupation), and thus on the implementation programme. It was discussed in our previous work [36], that conflict is to all intents and purposes inherent within large multi-partner enterprise system implementations, and as such is almost always going to occur at some point during the lifecycle of the

programme-wide implementation, or within one or more of its constituent project teams. Our ABM has been designed to model this case. It allows us to simulate the development and propagation of conflict throughout the social network of the workplace relationships within the individual project teams and the wider programme as a whole.

4. User Experiences of FLAME when Modelling the Case Study

The ABM [41] was designed and developed to reflect findings from our previous qualitative studies [36, 42]. In the subsections below, we discuss our experiences using FLAME following the design, development, and simulation-based experimentation of an ABM of conflict development and propagation within our case study. Our discussion focuses on four main areas: the ease of model design having made the decision to use FLAME; the ease of model development using FLAME; constraints due to the underlying architecture of FLAME; and performance of FLAME in running simulations.

4.1. Ease of Model Design

As discussed above, the FLAME simulation framework utilizes the concept of communicating stream X-Machines to define the logical entities and the rule-based logic of their interactions within the agent-based model. The conceptual framework [36, 42], acted as the functional specification for our ABM, which in turn was used as the basis for developing our technical specification. When developing the technical specification, we were cognizant of the underlying architecture of FLAME, and made explicit reference to: the C programming language and the XML markup language, which were used for coding agent interaction functionality and defining agents; the communicating stream X-Machine architecture for defining agents and their associated behaviours and dynamics; along with the use of the centralized message board to facilitate communication between agents.

Briefly, our technical specification consisted of a two-dimensional, undirected network of 159 agents that represented the individual team members from the case study. Communication between individual members followed the rules specified in the adjacency matrix and resulting social network map (figure 4), and this communication within the ABM is modelled as a process with no velocity (i.e., link between two agents, but no speed). Specification of the agents was straightforward because the communicating stream

X-Machine architecture facilitates easy compliance to an object-oriented approach to design. In addition, FLAME is flexible with respect to defining the physical boundary of the model, which allowed us to design a 2-Dimensional physical environment for our ABM, and the use of Cartesian co-ordinates to position individual agents within the environment. This allowed us to group agents together in 2-Dimensional space to represent the clustering of resources into their respective project team, and to disperse individual teams within an open-plan office, another office in the building, or potentially in another location.

System-level behaviours corresponded to those depicted in our Rich Picture (figure 3), with the main agent attributes that facilitate simulation of conflict development and propagation relating to: project team (e.g., Programme, HR, Financials, etc.); Organization (e.g., Client, Vendor, Consulting Firm); Role Type (e.g., Management, Functional, Technical, Training); Conflict Quotient for each type of conflict; Stress Quotient for each type of conflict; and Formal Authority Quotient, which indicates how much formal authority/power the agent has over others. With respect to communication between agents, although FLAME utilizes a centralized message board, we were able to design restrictions into the communication mechanism to ensure agent communication corresponded to the workplace social interactions, as defined within the adjacency matrix from our case study (i.e., the 972 workplace relationships between agents). This was facilitated through adding an attribute to the agent definition that explicitly defined the other agents that it is able to communicate and interact with.

One point to note is that when diagrammatically modelling an X-Machine as part of the technical specification, the X-Machine state is associated with the *computational* system state transitions and does not correlate to the *social* (domain specific) states. These internal X-Machine state transitions are achieved through the use of transition functions, which encode the logic that changes the agents memory, and communicates with other agents through the use of messages sent to (using output port) and received from (using input port) the centralized communication matrix. For those familiar with UML, an ABM developed using X-Machines can be specified (from a system behavioural perspective) as a set of connected state transition diagrams.

4.2. Ease of Model Development

For those comfortable with integrated development environments (to write code), the command line terminal (to run simulations and submit batch

jobs to HPC architecture), and the use of shell scripts (to submit jobs to a server, link to scripts for post-processing of output files, and link to analysis scripts [such as R or Matlab] to generate descriptive statistics and graphs), the FLAME simulation framework is an intuitive and very friendly tool for developing agent-based models using XML and C. Unfortunately, the converse of this holds for the novice modeller, and we conjecture that a more user friendly agent-based modelling framework that uses a graphical user interface, such as NetLogo, would be suitable as a first step to developing competence in model development and simulation-based experimentation.

FLAME provides a simulation engine that parses the ABM specification files (XML agent definition files and C functions files), manages the execution of simulations (on either a single node or over parallel architecture), and manages potential interactions between individual X-Machine agents. Within FLAME simulation runs, the notion of time is modelled as discrete *time-steps*. Each time-step therefore involves the individual X-Machines to iterate through their internal computational states, which may result in updates to their real-world *social* states following interactions with either the environment or other X-Machine agents. Furthermore, for those familiar with object-oriented design and development, FLAME, being underpinned by the concept of communicating stream X-Machines, allows the various functions within the ABM to be constructed in a modular fashion. Consequently, FLAME ensures that each type of X-Machine agent can be designed in an object-oriented way, which allows their construction and modification to be performed without the risk of affecting the functionality of other agent types as long as a standard interface is used for communication and interactions between agents.

The underlying process related to the technical architecture of FLAME is the requirement for synchronization, through a simulation-level time-step after all agents within the simulation run have completed one internal computational transition function. The effect of this technical processing constraint, is that the end and start of computational transition functions, incorporate the ability to synchronize communication and internal computational state transitions. With this in mind, care is required to ensure that agent behaviours do not contain any loops, which would incur the risk of undecideability problems, and that the interactions between agents (through use of the centralized message board) does not result in causal loops, which could impact upon the step-by-step updating of the agent internal states.

Coakley openly acknowledged during his design and development of the

FLAME simulation framework that *"when communicating X-Machines are used to represent agents in an [ABM], communication is [often] restricted to interactions with neighbouring agents that are [located] close to one another"* [20]. Within our social system model, there are two broad categories of messages used within the simulation runs to communicate an X-Machine agents' current location and their potential to initiate or be part of a social interaction. The location message is sent from individual X-Machines to the communication matrix to broadcast their current location so that other X-Machine agents can calculate whether they are within the appropriate spatial range for initiating an interaction, or alternatively to determine whether they are connected within the social network. With specific reference to the latter, the agents will elucidate whether they are connected (as defined in the adjacency matrix), and once [a] suitable agent(s) is/are identified, messaging to the centralized communication matrix would then be used to initiate interactions between the relevant agents.

Finally, one point to be aware of, is that although development of an ABM is one of the principal essential activities, in isolation, it will not enable simulation-based experimentation. In order to run simulations and analyze the resultant data, a number of additional computational tools need to be developed (termed *instruments*). For our research, we developed the following instrumentation tools alongside the development of the ABM itself: Unix shell (specifically, BASH) and Ruby scripts to facilitate a semi-automated submission of simulation runs to the HPC resources; Python scripts to facilitate a semi-automated mechanism to generate the initial starting parameters files (using XML templates) for initiating simulation runs; scripts for the processing and transformation of XML output files to CSV files (Python and Matlab scripts); scripts to automatically generate graphs (in Matlab and R) for analyzing the data; a visualization tool to provide a graphical and animated view of the simulation run's system dynamics over time (separate package provided by the FLAME development team, that can be configured to the ABM); and various statistical analysis scripts (developed in Matlab and Python) to analyze the simulation output data for statistical significance using the Kolmogorov-Smirnov test [43], and effect magnitude using the A-Test [44]. These instrumentation tools (apart from the FLAME visualizer) were developed using the appropriate scripting/programming language, so in order to ensure that ABMs developed in FLAME can be robustly tested and analyzed, the developer requires a working knowledge of various statistical techniques along with competence in various scripting/programming

languages. With that in mind, we do not believe that FLAME is appropriate for the novice modeller or subject matter expert.

4.3. Constraints due to Underlying Architecture

With FLAME being designed and developed from the outset to utilize parallel processing hardware architectures, the notion of a *global mutable parameter* unfortunately does not exist within the simulation framework. This is because such functionality would introduce issues around concurrency control and potentially lead to dead-locking. For instance, an individual compute node within the HPC architecture may be processing the simulation data for a particular X-Machine agent that is dependent on the current value of a global parameter, but the global parameter is currently being processed by another compute node. To mitigate risks around using outdated parameter values, the compute nodes would be required to *wait* until they confirmed that the global parameter was not being processed. When amplified up to the level of the overall simulation run, the cumulative *waiting* would incur a significant overhead regarding the Wall-Clock time of simulation runs, which would significantly offset the benefits of parallel processing. Consequently, we were unable to utilize a global counter to keep track of system-level quantities or to set system-level chracteristics, such as a pseudo-random number generator seed value.

4.3.1. Setting the Pseudo-Random Number Generator Seed

Pseudo-Random Number Generators (PRNGs) allow us to introduce probabilistic behaviour into our computational models, such as the interactions between X-Machine agents, or between the agents and the system environment. During the calibration exercise, and later more formally during the verification process, we ensure that the computational model is not over-tuned through running multiple replicate simulations that utilize different PRNG seed values and performing statistical analysis to ascertain the variance in simulation output data. The principle behind explicitly setting the seed value for a PRNG is that you set once, and use multiple times [45]. A significant constraint that we discovered with FLAME is that it does not include a mechanism for you to do this very easily, because mutable parameter values can only be defined and set within the definition of an agents' logic (the C functions file). As discussed in the previous paragraph, the design decisions taken when FLAME was first developed means there is no opportunity to set a global constant within the simulation environment setup of a

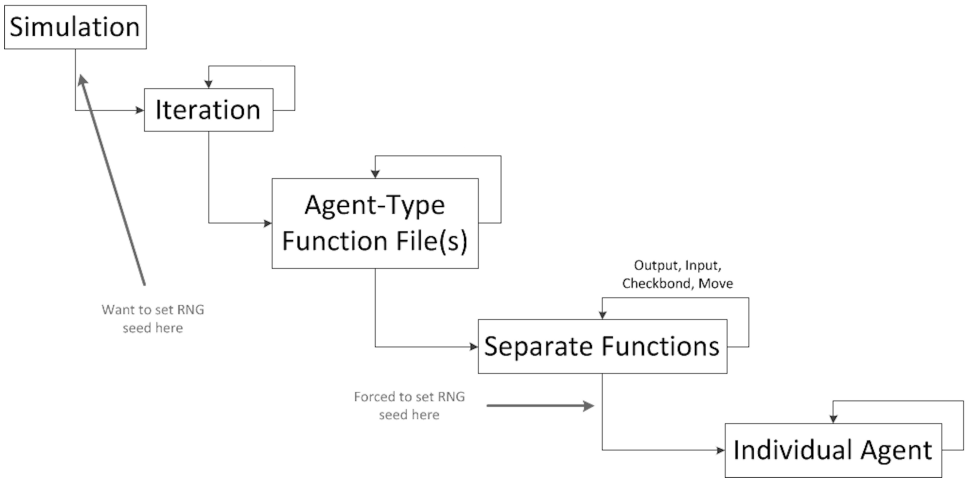

Figure 5: The levels within a simulation run at which the PRNG seed value can be set when using FLAME. Although the principle of using PRNG seed values is that you set once per simulation and use multiple times, FLAME's explicit exclusion of global mutable parameters means that this is not straightforward. Due to FLAME requiring all functionality to be written at the level of agents, this provides us with the ability to experiment to see whether any of the levels allow us to develop a technical workaround, for instance through setting at the level of agent-type, where any function would be called once per agent-type per iteration, or at the level of individual agents, where any function would be called once for each agent and for each iteration.

simulation run due to the inefficiencies involved with it propagating across the multiple nodes within the HPC architecture. With that in mind, not only are global variables, such as total counts, unable to be updated, but the PRNG seed value cannot be set at the global level. This introduces the constraint of either not being able to set your own PRNG seed value, and therefore being forced to use the system clock to derive the seed value, or to develop a technical workaround where the PRNG seed value is explicitly set within an individual agent's definition and logic (see figure 5).

When setting the PRNG seed value at the level of agent-type, you unfortunately encounter the issue of the seed value being reset during each simulation time-step, and thus do not gain the full benefit of using a PRNG to simulate stochastic behaviours within the computaional model. Through explicitly setting the PRNG seed value a list of *random numbers* is generated, albeit in a determinstic manner so that you can reproduce simulation runs by using the same seed value (hence the use of the prefix *pseudo*), to be used within probabilistic functions [46]. The resetting of the PRNG seed with the

same value and at the level of agent-type, results in the same deterministic list of pseudo-random numbers being generated as per the previous simulation time-step, which reduces the benefits of using a PRNG. This can be succinctly illustrated within our case study: if you set the seed value at the level of an individual agent within our ABM that contains 159 agents (e.g., separate programme team members) over a 2,000 time-step simulation, you in effect repeat the resetting of the seed value 318,000 times. Likewise, if you set the seed value at the level of agent-type, you in effect repeat the resetting of the seed value 2,000 times (assuming a single high-level agent type of *team member* and 2,000 time-step simulation run). Although the overall effects at this level are orders of magnitude smaller (e.g., far fewer resets of the PRNG seed value), it still significantly affects our ability to incorporate stochasticity into the ABM and associated simulation runs.

One approach for incorporating stochasticity into ABMs developed using FLAME, and therefore gaining variance between replicate simulations is to run simulations using the built-in *Production mode* within FLAME, which derives the seed value from the system clock. Unfortunately, this approach does not enable exact reproduction of individual simulation runs because the system clock is continuously updating with the progression of time, which seriously impacts our ability to repeat simulation-based experiments that have interesting dynamics, or to help resolve issues during the debugging activities in model development. Fortunately, a technical workaround was identified that resolved this issue, which involved the creation of a *dummy* agent whose sole purpose is to set the PRNG seed value in the first iteration of an individual simulation run. This utilized an agent-level counter (to count the simulation time-step), and logic to *only* set the PRNG seed value when the counter equals zero. In addition, and to keep the simulator tidy of computational artefacts, the logic also ensures that this dummy agent is removed from the simulation once the simulation time-step counter reaches a pre-defined number.

4.3.2. Unable to Send Messages between Simulation Time-Steps and Need to Explicitly Set Memory Allocation

Additional constraints, although minor in relation to the above, are that a single simulation time-step is taken as a standalone run of a simulation, and that memory allocation for the agents and messages requires a continuous block size of memory. With respect to the single simulation time-step, this means that upon completion of all functions in that time-step, the global

message board is emptied of all messages generated by agents during that time-step. Simulation modellers therefore have to take this constraint into account when designing and developing the ABM, because messages cannot be sent between time-steps, so pertinent information (e.g., temporary parameter values or communication from specific neighbours) will need to be stored within the agent's memory (i.e., within an agent attribute). Conversely, with respect to the block size, this is important for parallelization when using MPI, because the simulator needs to know how to package up data that is sent across nodes, so the need to explicitly define the size of the agent memory and messages (both in bytes) is crucial.

4.4. Performance

During the calibration process, initial simulations were run using a desktop PC that had quad-core processor, 8GB RAM and used the Windows operating system. This meant that three separate simulation runs could comfortably be performed simultaneously in distributed mode across three out of the four cores in the processor. We found the performance of FLAME to be very promising during initial evaluation when running a single simulation on a single compute node across multiple hardware platforms. Both *CPU time* and *Wall-Clock time* were found to be linear following increases in the number of agents within a simulation, and also linear when the simulation length increased up to 50,000 time-steps (the maximum needed in our simulation), which both indicate that FLAME scales very well. However, further investigation identified that the rate-limiting processes and tasks associated with simulation runs was the Input/Output ([I/O], e.g., Read and Write) speed for writing and reading the large number of XML files that are created as part of the simulation output. The cause for this relates to the underlying design principles of FLAME, with its conceptual architecture being based on communicating stream X-Machines, which require the generation of individual XML files for each simulation time-step (these XML files contain parameter values and states for every X-Machine agent that is instantiated). Due to the output XML file for a specific simulation time-step, becoming the input XML file for the next simulation time-step, a performance bottleneck is encountered with respect to the speed of reading from and writing to, the storage disk.

For example, our ABM contained 159 separate agents that corresponded to the individual team members in the case study, and resulted in each XML output file being approximately 50kb in size. A 50,000 iteration simulation

run, would therefore generate 50,000 individual simulation time-step output files, that contain the parameter values and states associated with each of the 159 agents at the respective simulation time-step, which in our case approximated to 2.5Gb of output data per simulation run. Furthermore, following aleatory uncertainty analysis, we discovered that 75 replicates of each simulation run (using different PRNG seed values) are required to stabilize the median averages of system dynamics [47], which results in approximately 190Gb of simulation output being generated for each set of simulation runs. As such, complex ABMs that incorporate a large number of agents, can quickly generate significant volumes of simulation output data, which could significantly impact the ability to perform simulation-based experimentation if large capacity storage is not available on the computing hardware.

Our relatively simple ABM took approximately 25min to run on the Windows PC, along with an extra 10min to process and transform the XML simulation output files (using Python scripts) to CSV files, and then subsequently transforming these into a single CSV file that contained the median average dynamics of each agent over the duration of the simulation. Through using a pipeline of scripts in this way, we were able to ensure that the large volume of output data spent the minimum of time on the storage facility.

To confirm I/O rate-limiting characteristics of FLAME, we performed an identical set of simulation runs using the same Windows PC, but for this experiment, the output files were written to an external USB hard drive. We found the overall Wall-Clock time increased due to the flow of data across the USB port being slower than that to the onboard hard disk. We built upon these initial findings through using another desktop setup, which this time encompassed an Apple Mac Mini with 8GB RAM and Solid State Disk (SSD), along with a High-Performance Computing facility run by the Northern Eight Universities Consortium (N8) (based within the North of England). The Mac Mini with SSD was 28% faster (with respect to Wall-Clock time) than the Windows PC with internal disk, and 38% faster than that setup when using an external USB hard drive, resulting in the overall Wall-Clock time of a 50,000 time-step simulation run being reduced down to 18min. Likewise, the N8 HPC was 85% faster than the setup using Mac Mini with SSD, bringing the Wall-Clock time for simulation runs down to just under 3min (see figure 6). Access to fast storage (of suitable large capacity) is therefore crucial for any moderately complicated ABM that requires replicate simulations to be run in order to account for the aleatory uncertainty within the model.

An additional observation on performance relates to the communication

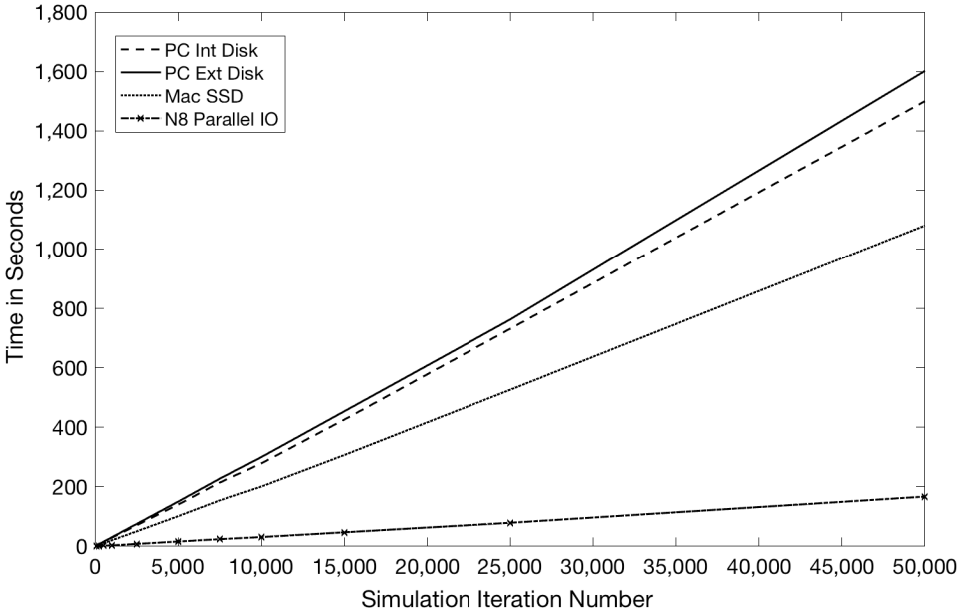

Figure 6: The performance of FLAME was found to be linear with respect to the number of iterations within a simulation run and the wall clock time. FLAME was found to be Input-Output rate-limiting by running a simulation on various hardware setups. For instance, we ran the simulation on: a Windows PC with internal HDD and on an external HDD, a Mac Mini with SSD, and a high-performance cluster with dedicated fast lustre storage.

between agents over parallel architectures, and the resultant sequence of messages being sent and read. Such communication dependencies between agents requires a synchronization block between the cores (on a laptop/desktop) or nodes (on a HPC) to ensure that messages arrive in time to meet the dependency, i.e., functions being performed in the specific time-step of the simulation do not try to read a message before the node/core has received it from another node/core. These synchronization blocks are a considerable time bottle neck, which introduce performance losses into simulations. As such, the fewer synchronization blocks introduced into a simulation, the greater the performance of the overall simulation run.

5. Discussion

Real-world social systems are complex, with sets of behaviours, characteristics and dynamics that emerge through the individual relationships that

function through time and space. One of the major benefits of the agent-based modelling paradigm, is the emphasis on three principal characteristics of complex systems: 1) system structures, 2) system dynamics, and 3) system control [3]. One of the advantages of the agent-based approach is that the simulations aim to replicate the dynamics of the real-world system, in order to ensure the validity of the underlying assumptions behind the computational model can be tested. However, in order for these ABMs to be successful in performing their role as scientific instruments that act as credible abstractions of complex social systems, it is crucial that the preferred modelling and simulation framework is able: to realistically represent system structure and dynamics; is modular, so that the ABM can be incrementally updated with new functionality, without the need to re-engineer the entire model; can expand with reference to the hierarchical-scale of the real-world system, e.g., individual team members, to discrete project teams, to the programme-wide network as a whole; and is amenable to a thorough validation and verification process, including stringent statistical analysis of simulation output data.

Within this study, we used FLAME to develop an ABM that was calibrated to the quantitative and qualitative data from our case study around the development and propagation of conflict within multi-partner enterprise system implementations. As discussed above, the FLAME modelling and simulation framework utilizes the conceptual and technical architectures associated with communicating stream X-Machines to facilitate agent-based models in a discrete-event manner (the simulation time-steps are the discrete events). It has been reported to provide very significant improvements in performance over more traditional ABM frameworks [48]. FLAME's overarching purpose is to deal with massive simulations, allowing for modelling abstraction levels that cater for large scope with respect to the real-world system of interest, and hundreds of thousands, to millions, of X-Machine agents. Through being designed and developed to comply with the MPI communication framework, FLAME code is also deployable on to parallel hardware platforms.

Our study has identified a number of strengths and weaknesses for using FLAME to model complex social systems. Firstly, with respect to the strengths, and for modellers who are familiar with the object-oriented approach to design of computational models, we have identified that FLAME is an intuitive framework for designing ABMs of complex social systems due to its template-driven approach. Similarly, for modellers who are comfortable with command line terminal, basic programming languages, and shell

scripts, we have identified that FLAME is an intuitive framework for developing ABMs of complex social systems. Conversely, we have shown that FLAME suffers with a number of constraints that introduced technical issues into the development process for our ABM. The four major constraints were: the lack of built-in functionality to set the PRNG seed value at the simulation level; the inability to utilize functionality associated with global mutable parameters; the inability to communicate between simulation time-steps/iterations; and the lack of instrumentation to analyze simulation output data, thus requiring the development of various statistical and data transformation scripts, which is fine for experienced modellers, but is a real constraint for novice modellers or subject matter experts.

These constraints are unfortunately a direct consequence of the design decisions taken during the initial creation of FLAME, because the underlying premise was that it would harness the power of parallel processing to run individual simulations across HPC architectures. A number of technical challenges were introduced by these constraints, but fortunately, we were able to develop workarounds to resolve them. As such, we want to inform future users of the FLAME simulation framework that they need to be cognizant of these constraints, especially due to the fact that it has been proposed as suitable for subject matter experts within the real-world domain of interest, who may have limited experience and/or competence in computational modelling or programming. With this in mind, we advocate that this message be moderated, because subject matter experts may not have the technical competence or modelling experience to fully investigate, analyze, diagnose and resolve the full variety of problems and constraints that we have experienced during this study.

Furthermore, and of more significance is that we discovered a major limitation when using FLAME, is that models that incorporate large numbers of individual agents are computationally expensive with respect to Wall-Clock time, the I/O load, the need for HPC architecture to run multiple replications in distributed mode, and the consequent very large size of output simulation data. This *computational expense* is compounded when the ABM is increased in scale to reproduce results consistent with system-level dynamics from the real-world domain. Due to FLAME's communicating stream X-Machine architecture, which relies on input and output streams of messages to facilitate communication, we unexpectedly discovered that comparatively small simulations (with respect to total X-Machine agent numbers, the complexity of interaction logic, and the total time-steps within the simulation) could gen-

erate a considerable number of XML data files, which directly correlates to the volume of simulation output data produced. The Wall-Clock time for single simulation runs was found during diagnostic tests, to grow linearly in accordance with the number of time-steps within simulation runs. Diagnostic tests also indicated the Input-Output rate-limited nature of FLAME (as opposed to being rate-limited through the Central Processing Unit), which is due to separate XML output files being generated for each time-step within simulation runs.

To put this into further context, the communicating stream X-Machine nature of FLAME requires that individual XML output files are generated at the end of each time-step to record the parameter values for internal computational and simulated real-world states along with internal memory values for each X-Machine agent; these are then used as the XML input file at the beginning of the next time-step to set the corresponding starting parameter values, states, and memory values for the next time-step of the simulation run. A bottleneck is therefore experienced regarding computational performance, which results from the speed of reading from and writing to, the onboard or peripheral storage disk. The results from aleatory uncertainty analysis further compounded this performance issue by identifying that a minimum of 75 simulation replicates were required to achieve stable median dynamics, which resulted in approximately 3,750,000 total XML output files, which as disucssed, amounts to 190GB for our relatively small ABM.

There can often exist a delicate balance with respect to computational efficiency and performance during aleatory uncertainty analysis, which is introduced from the desire to achieve stable simulation results by using a high number of replicates for calculating the median average simulation results, whilst being cognizant of the computational resources required (e.g., Wall-Clock time, number of CPU cores on desktop or nodes on HPC, and access to fast storage disk). The number of simulation replicates chosen to calculate the median average dynamics is therefore usually a compromise between minimizing the impact of aleatory uncertainty versus the acceptable costs in computational resources for the project. Our findings therefore indicate that *average, everyday* desktop computing resources (such as a Windows PC or Apple Mac Mini) are unsuitable for running simulations that require large numbers of replicates to achieve stable average dynamics to act as a baseline for simulation-based experimentation. We therefore conjecture that access to very fast multi-core desktops with significant storage capacity (such as fast SSD raid capabilities), or to HPC architecture is critical for ensuring that

our ABM of conflict development and propagation within a complex social network can be used for simulation-based experimentation.

6. Conclusion

The above discussion has focused on our contributions to the computational modelling community through out user experiences when using FLAME to model a complex dynamical social system. Specifically, the discussion has focused on the strengths and weaknesses that we identified when using FLAME to model the development and propagation of conflict within the social network of large multi-partner enterprise system implementations. We found the design of our model to be straightforward due to FLAME's underlying architecture using communicating stream X-Machines. The object-oriented nature of communicating stream X-Machines, which are in effect, computational instantiations of UML state transition diagrams, makes the design and subsequent development relatively intuitive. In addition, the use of XML to define agents and C to code the agent communication and interaction rules, allows for a truly modular approach to ABM development. Conversely, however, the limitations of FLAME are also due to the underlying conceptual and technical architecture of FLAME. We discovered four main constraints relating to the lack of in-built functionality to set PRNG seed values, the inability to use global mutable parameters, the inability to communicate between time-steps, and the lack of instrumentation to analyze simulation output data. In addition, we identified a significant limitation with respect to performance, which is a direct consequence of the communicating stream X-Machine architecture, in that the output file for a simulation time-step acts as the input file for the subsequent time-step. Our analysis indicated that this not only leads FLAME to be I/O rate-limiting, but also means that large simulations (with respect to either total number of agents or total number of simulation time-steps) quickly generate prohibitively large amounts of simulation output data. We were lucky that our ABM of the case study is relatively simple and had only 159 agents within the social network, but even still, following aleatory uncertainty analysis we were required to run 75 replicates of simulations in order to mitigate the effects of aleatory uncertainty, and produced 190GB. As such, we conjecture that for large, complex, ABMs, the large volumes of simulation output data generated by FLAME may mean that it is rejected as a potential simulation framework if access to very large capacity and high-speed storage capabilities cannot be achieved.

To conclude, we believe FLAME is an excellent choice for experienced modellers, who will be able to fully harness the capabilities that it has to offer, and also be competent in diagnosing and solving any limitations that are encountered. Conversely, because FLAME requires considerable development of instrumentation tools, along with development of statistical analysis scripts, we believe that it is not suitable for the novice modeller, who may be better suited to using a graphical user interface driven framework until their experience with modelling and competence in programming increases. In our opinion, FLAME's major strength is its flexibility, in that once the model definition (along with any technical workarounds) has been developed using the XML and C templates (e.g., the social network topology, communication rules, rules for conflict development, and functionality to explicitly set the PRNG seed value), the augmentation of the ABM with new functionality is comparatively easy due to the template-driven nature of FLAME facilitating its modular approach to design and development. In addition, we have found FLAME to be excellent for quickly developing ABMs of complex dynamical systems in both our case study presented here, and indeed in previous work into complex dynamical biological systems [21]. We do however believe that it requires significant computational modelling skills in order to develop workarounds to some of the constraints that have been imposed due to the underlying technical architecture, and to develop a pipeline of scripts (e.g., Ruby, Unix Shell, MS DOS, and Python) to semi-automate the submission of jobs to a HPC and to transform and analyze simulation output data to reduce the storage load when large numbers of replicate simulations are required to account for the aleatory uncertainty within the model. Finally, we are aware that a version of FLAME has been developed that harnesses the parallel processing power of Graphics Processing Units (GPU) [49], which has recently been extended [50]), and could prove beneficial, because a number of the limitations that we discovered (e.g., no global mutable parameter and the I/O resource intensiveness) may be resolved through the difference in processing architecture. One final point to highlight on this however, is that FLAME GPU, being based on CUDA, requires NVIDIA graphics cards, which could prove problematic for those developers who are using Apple Mac computers because of the current issues with drivers and Mac Mojave/Catalina operability, and the need to use eGPU and various scripts to get the thunderbolt connectivity working with older Macs - we are aware that newer Macs have Thunderbolt 3 connectivity and that Mac OS High Sierra supports NVIDIA drivers. To this end, we are in the process of trialling FLAME GPU on such

a setup.

Acknowledgement

This work was in part funded by a Lancaster University Early Career Small Grant and a Management and Business Development Fellowship awarded jointly by the Society for the Advancement of Management Studies, the United Kingdom Commission for Employment and Skills, and the Economic and Social Research Council (SAMS-UKCES-ESRC) with Grant No. ES/L002612/1. The funders had no involvement in study design; development of the underlying ABM, its analysis and interpretation of results; in the writing of this manuscript, or the decision to submit this manuscript for publication.

References

[1] C. Cioffi-Revilla, Computational social science, WILEY Interdisciplinary Reviews: Computational Statistics 2 (3) (2010) 259–271.

[2] G. Cordasco, V. Scarano, C. Spagnuolo, Distributed MASON: A scalable distributed multi-agent simulation environment, Simulation Modelling Practice and Theory 89 (2018) 15–34.

[3] T. Ideker, T. Galitski, L. Hood, A new approach to decoding life: Systems biology, Annual Review of Genomics and Human Genetics 2 (2001) 343–372.

[4] R. A. Williams, Lessons learned on development and application of agent-based models of complex dynamical systems, Simulation Modelling Practice and Theory 83 (2018) 201–212.

[5] M. Carillo, G. Cordasco, F. Serrapica, V. Scarano, C. Spagnuolo, P. Szufel, Distributed simulation optimization and paramter exploration framework for the cloud, Simulation Modelling Practice and Theory 83 (2018) 108–123.

[6] J. Epstein, Agent-based computational models and generative social science, Complexity 4 (5) (1999) 41–60.

[7] S. Stepney, F. A. C. Polack, Engineering Simulations as Scientific Instruments: A Pattern Language, Springer, London, UK, 2018.

[8] C. M. Macal, M. J. North, Tutorial on agent-based modeling and simulation, in: M. E. Kuhl, N. M. Steiger, F. B. Armstrong, J. A. Jones (Eds.), Winter Simulation Conference, Orlando, Florida, 2005, pp. 2–15.

[9] N. R. Jennings, On agent-based software engineering, Artificial Intelligence 117 (2000) 277–296.

[10] E. Bonabeau, Agent-based modelling: Methods and techniques for simulating human systems, Proceedings of the National Academy of Sciences 99 (s3) (2002) 7280–7287.

[11] A. A. Dodson, S. Stepney, E. Uprichard, L. Caves, Using the CoSMoS approach to study schelling's bounded neighbourhood model, in: S. Stepney, P. S. Andrews (Eds.), Proceedings of the 2004 Workshop on Complex Systems Modelling and Simulation, Luniver Press, New York, NY, USA, 2014, pp. 1–12.

[12] P. Garnett, A tipping point in 300 years of banking? a conceptual simulation of the british banking system, Natural Computation 14 (2015) 25–37.

[13] V. Sahasrabudhe, S. Kanungo, R. Iyer, Understanding the impact of communication technologies on virtual team performance: An agent-based simulation model, in: S. Jain, R. R. Creasey, J. Himmelspach, K. P. White, M. Fu (Eds.), Proceedings of the 2011 Winter Simulation Conference, IEEE, Phoenix, Arizona, USA, 2011, pp. 321–332.

[14] C. S. M. Currie, J. W. Fowler, K. Kotiadis, T. Monks, B. S. Onggo, D. A. Robertson, A. A. Tako, How simulation modelling can help reduce he impact of COVID-19, Journal of Simulation Advanced Online Publication (2020).

[15] M. E. Csete, J. C. Doyle, Reverse engineering of biological complexity, Science 295 (2002) 1664–1669.

[16] C. Nikolai, G. Madey, Tools of the trade: A survey of various agent based modeling platforms, Journal of Artificial Societies and Social Simulation 12 (2) (2009) 2.

[17] S. Luke, C. Cioffi-Revilla, L. Panait, K. Sullivan, G. Balan, MASON: A multi-agent simulation environment, Simulation 81 (2005) 517–527.

[18] ROAD, Repast home page,, repast Orgainzation for Architecture and Design, Chicago, IL: available at http://repast.sourceforge.net (2005).

[19] U. Wilensky, Netlogo, the center for connected learning and computer-based modeling, northwestern University, Evanston, IL, http://ccl.northwestern.edu/netlogo/ (1999).

[20] S. Coakley, R. Smallwood, M. Holcombe, Using X-Machines as a formal basis for describing agents in agent-based modelling, Simulation Series 38 (2) (2006) 33–40.

[21] R. A. Williams, J. Timmis, E. E. Qwarnstrom, Investigating IKK dynamics in the NF-κB signalling pathway using X-machines, in: Proceedings of the IEEE 2017 Congress on Evolutionary Computation, IEEE Xplore, Donastia-San Sebastian, Spain, 2017.

[22] M. Holcombe, S. C. Coakley, M. Kiran, S. Chin, C. Greenhough, D. Worth, S. Cincotti, M. Raberto, A. Teglio, C. Deissenberg, S. van der Hoog, H. Dawid, S. Gemkow, P. Harting, M. Neugart, Large-scale modeling of economic systems, Complex Systems 22 (2013) 175–191.

[23] B. Rachid, T. Mohamed, M. A. Khouaja, An agent based modeling approach in the strategic human resource management, including endogenous and exogenous factors, Simulation Modelling Practice and Theory 88 (2018) 32–47.

[24] P. Heywood, P. Richmond, S. Maddock, Road network simulation using flame gpu, in: S. Hunold, A. Costan, D. Gimenez, A. Iosup, L. Ricci, M. E. Gomez Requena, V. Scarano, A. L. Varbanescu, S. L. Scott, S. Lankes, J. Weidendorfer, M. Alexander (Eds.), Proceedings of the 21st International Conference on Parallel and Distributed Computing, Vol. 9523 of Lecture Notes in Computer Science, Springer, Vienna, Austria, 2015, pp. 430–441.

[25] I. Foster, Designing and Building Parallel Programs, Addison Wesley, 1995, Ch. Message Passing Interface.

[26] S. Coakley, M. Kiran, FLAME User Manual, University of Sheffield, Sheffield, UK (October 2009).

[27] S. Eilenberg, Automata Languages and Machines, Vol. A, Academic Press, New York, NY, USA, 1974.

[28] M. Holcombe, X-machines as a basis for dynamic system specification, Software Engineering Journal (69-76) (1988).

[29] E. Kehris, G. Eleftherakis, P. Kefalas, Using X-machines to model and test discrete event simulation programs, Systems and Control: Theory and Applications (2000) 163–168.

[30] I. Stamatopoulou, P. Kefalas, M. Gheorghe, Modelling the dynamics structure of biological state-based systems, BioSystems 87 (2007) 142–149.

[31] M. L. Palmer, R. A. Williams, D. Gatherer, Rosen's (M,R) system as an X-machine, Journal of Theoretical Biology 408 (2016) 97–104.

[32] L. Wylie, ERP: A vision of the next-generation MRP II, Scenario s-300-339, Gartner Group (April 1990).

[33] F. R. Jacobs, F. C. T. Weston, Enterprise Resource Planning (ERP) - A brief history, Journal of Operations Management 25 (2) (2007) 357–363.

[34] S. F. King, T. F. Burgess, Beyond critical success factors: A dynamic model of enterprise system innovation, International Journal of Information Management 26 (1) (2006) 86–97.

[35] S. J. Whitty, H. Maylor, And then came complex project management (revised), International Journal of Project Management 27 (3) (2009) 304–310.

[36] R. A. Williams, Conflict propagation within large technology and software engineering programmes: A multi-partner enterprise system implementation as case study, IEEE Access 7 (1) (2019) 167696–167713.

[37] K. Boulding, Conflict and Defense, Harper and Row, New York, NY, USA, 1963.

[38] K. A. Jehn, A multimethod examination of the benefits and detriments of intragroup conflict, Administrative Science Quarterly 40 (2) (1995) 256–282.

[39] K. A. Jehn, E. A. Mannix, The dynamic nature of conflict: A longitudinal study of intragroup conflict and group performance, Academy of Management Journal 44 (2) (2001) 238–251.

[40] N. Gamero, V. Gonzalez-Roma, J. M. Peiro, The influence of intra-team conflict on work teams' affective climate: A longitudinal study, Journal of Occupational and Organizational Psychology 81 (1) (2008) 47–69.

[41] R. A. Williams, Modelling conflict within the social networks of large multi-vendor software projects using communicating stream X-Machines, in: Proceedings of the European Conference on Artificial Life, MIT Press, Cambridge, MA, USA, 2015, p. 79.

[42] R. A. Williams, Cybernetics of conflict within multi-partner technology and software engineering programmes, IEEE Access 8 (2020) 94994–95018.

[43] F. J. Massey, The Kolmogorov-Smirnov test for goodness of fit, Journal of the American Statistical Association 46 (253) (1951) 68–78.

[44] A. Vargha, H. D. Delaney, A critique and improvement of the CL common language effect size statistics of McGraw and Wong, Journal of Educational and Behavioural Statistics 25 (2000) 101–132.

[45] S. C. Barry, How much impact does the choice of a random number generator really have?, International Journal of Geographical Information Science 25 (4) (2011) 523–530.

[46] K. P. van Niel, S. W. Laffan, Gambling with randomness: The use of pseudo-random number generators in GIS, International Journal of Geographical Information Science 17 (1) (2003) 49–68.

[47] M. Read, P. S. Andrews, J. Timmis, V. Kumar, Techniques for grounding agent-based simulations in the real domain: A case study in experimental autoimmune encephalomyelitis, Mathematical and Computer Modelling of Dynamical Systems 18 (1) (2012) 67–86.

[48] M. Kiran, P. Richmond, M. Holcombe, L. S. Chin, D. Worth, C. Greenough, FLAME: Simulating large populations of agents on parallel hardware architectures, in: Proceedings of the 9th International Conference on Autonomous Agents and Multiagent Systems, Vol. 1, IFAAMAS, Toronto, Canada, 2010, pp. 1633–1636.

[49] P. Richmond, S. Coakley, D. Romano, Cellular level agent-based modelling on the graphics processing unit, in: International Workshop on High-Performance Computational Systems Biology (HiBi'09), IEEE, Trento, Italy, 2009, pp. 43–50.

[50] M. K. Chimeh, P. Richmond, Simulating heterogeneous behaviours in complex systems on GPUs, Simulation Modelling Practice and Theory 83 (2018) 3–17.